# Отчет к лабораторной работе №13

## Common information

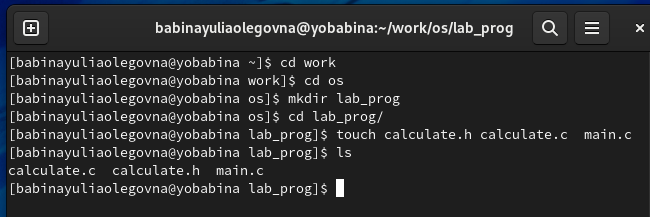
discipline: Операционные системы  
author: Бабина Юлия Олеговна  
group: НПМбд-01-21

## Цель работы

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

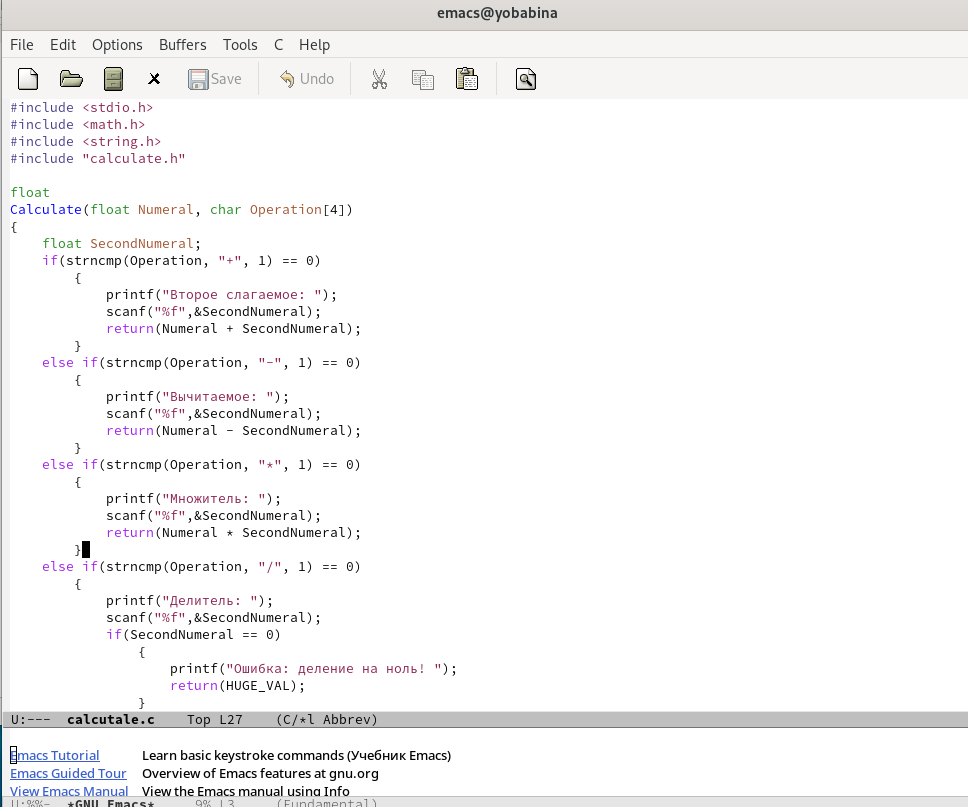
## Ход работы

В домашнем каталоге создаем подкаталог ~/work/os/lab\_prog и в нем уже создаем три файла: calcutale.h, calcutale.c, main.c. Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится.

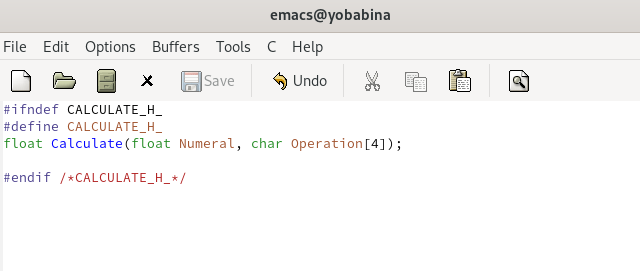


создание каталога и файла

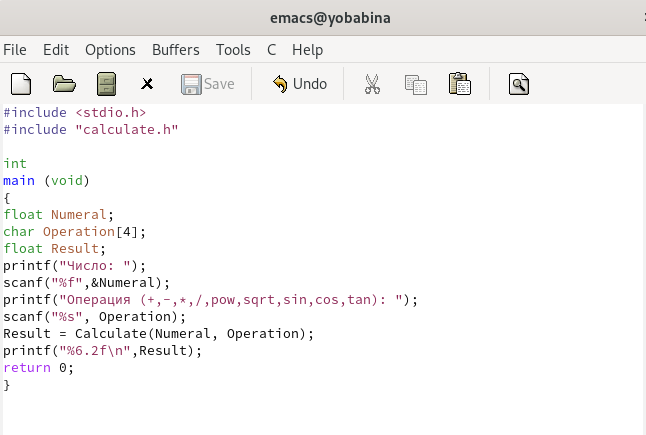
В созданных файлах напишем программы для работы калькулятора, которые нам предоставили.



файл calcutale.c

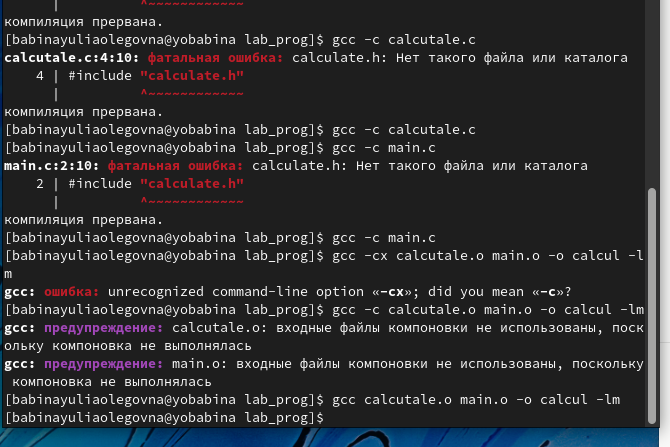


файл calcutale.h



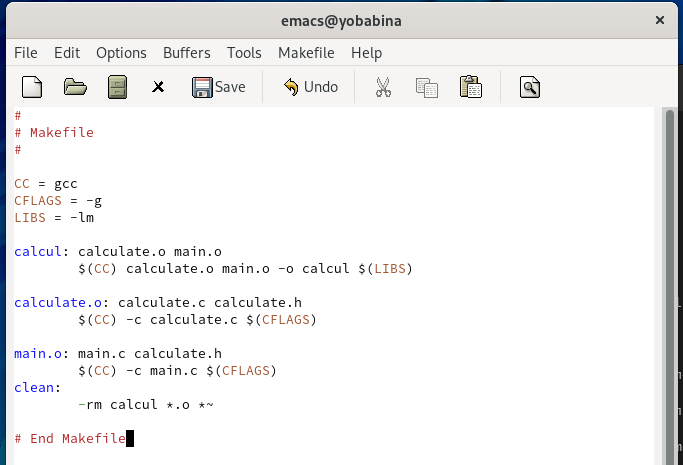
файл main.c

Выполним компиляцию программы посредством gcc:



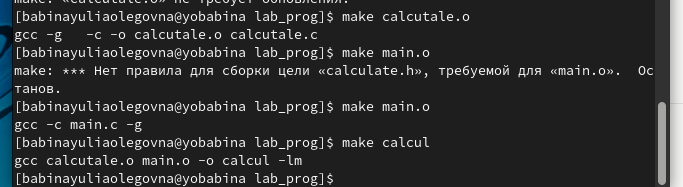
компиляция

Создадим Makefile со следующим содержанием:



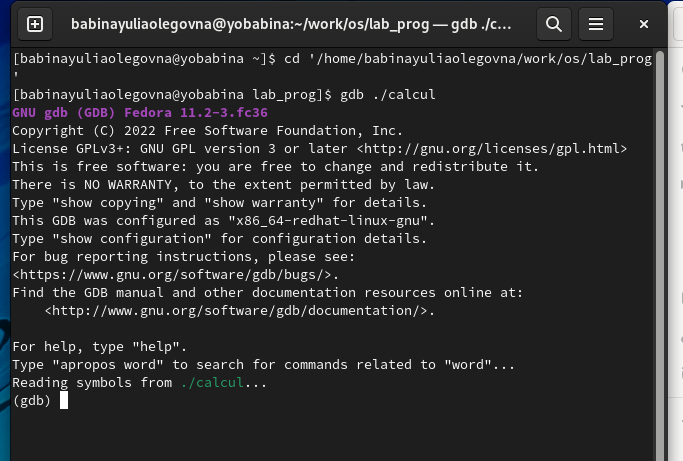
файл makefile

Пересоберем проект с помощью данного файла.

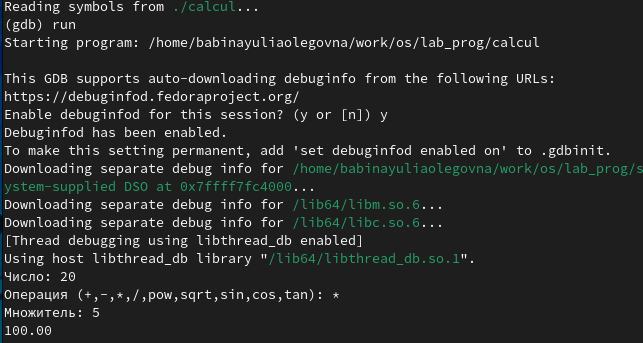


сборка при помощи файла make

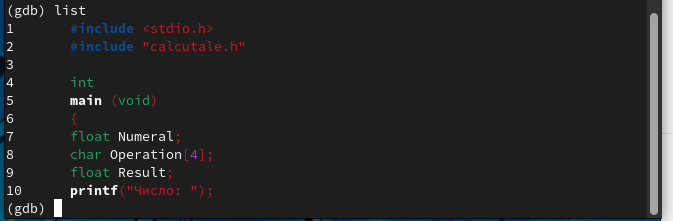
С помощью команды gdb выпполним отладку программы calcul.



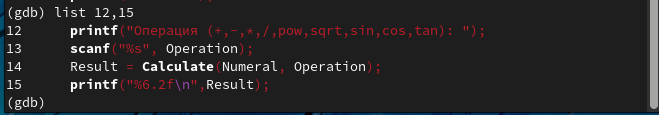
запуск команды gdb./calcul



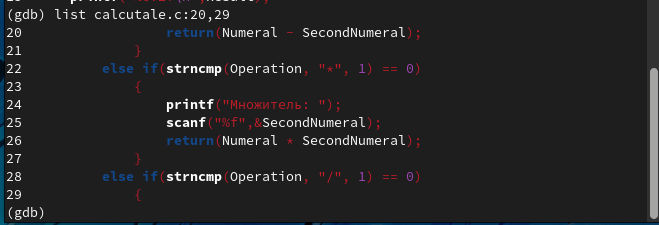
запуск команды run



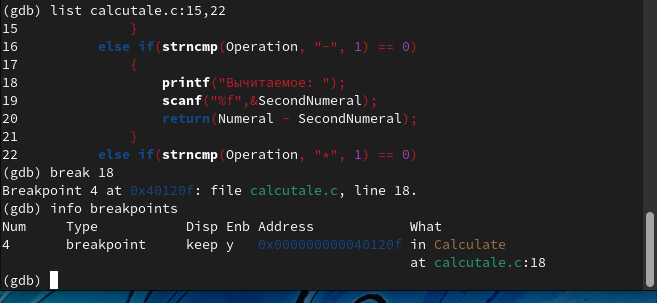
использование команды list



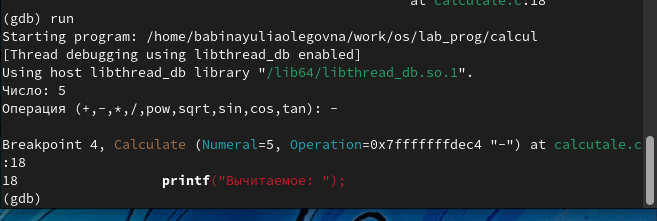
использование команды list



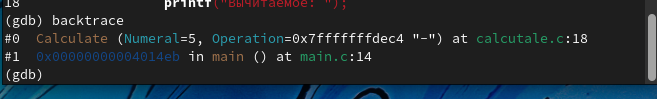
использование команды list



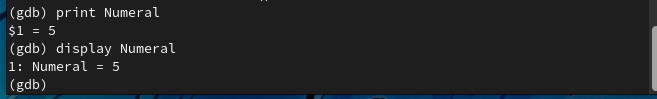
использование команды list



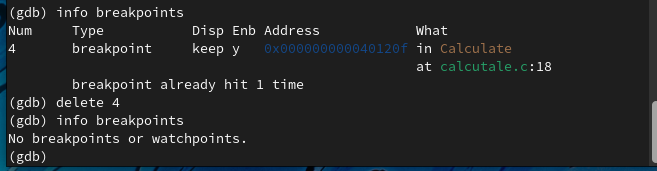
установка точки установа



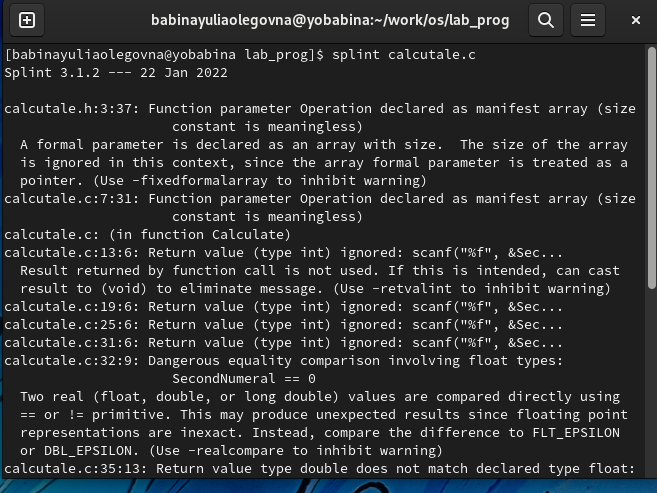
команда backtrace



проверка работы

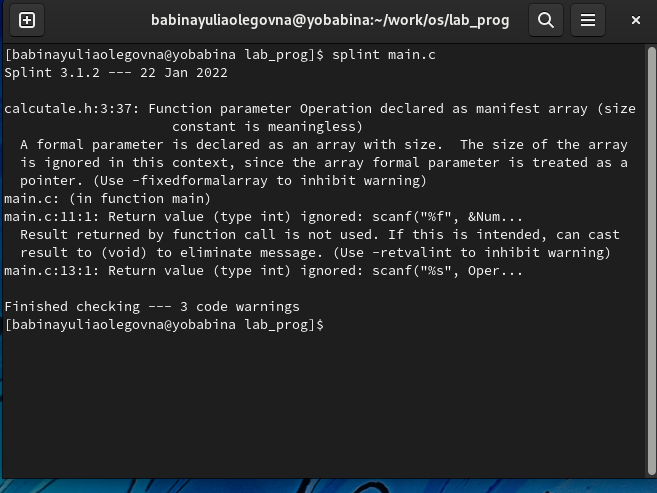


удаление точки останова



удаление точки останова

В конце с помощью утилиты split попробуем проанализировать коды файлов calcutale.c и main.c



утилита split

## Ответы на контрольные вопросы

### *Вопрос 1*

Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.

### *Вопрос 2*

Процесс разработки программного обеспечения обычно разделяется на следующие этапы: планирование, включающее сбор и анализ требований кфункционалу и другим характеристикам разрабатываемого приложения; проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования; непосредственная разработка приложения: кодирование − по сути создание исходного текста программы (возможно в нескольких вариантах); анализ разработанного кода; сборка, компиляция и разработка исполняемого модуля; тестирование и отладка, сохранение произведённых изменений; документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

### *Вопрос 3*

Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке С, файлы с расширением .cc или .C − как файлы на языке C++, а файлы c расширением .o считаются объектными. Например, в команде «gcc -c main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль − файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».

### *Вопрос 4*

Основное назначение компилятора языка Си в UNIX заключается вкомпиляции всей программы и получении исполняемого файла/модуля.

### *Вопрос 5*

Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

### *Вопрос 6*

Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае Makefile имеет следующий синтаксис: … : … <команда 1> … Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды − собственно действия, которые необходимо выполнить для достижения цели. Общий синтаксис Makefile имеет вид: target1 [target2…]:[:] [dependment1…] [(tab)commands] [#commentary] [(tab)commands] [#commentary] Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться водной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках.

### *Вопрос 7*

Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора gcc: gcc -c file.c -g После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdb file.o

### *Вопрос 8*

Основные команды отладчика gdb: backtrace − вывод на экран пути к текущей точке останова (по сути вывод − названий всех функций) break − установить точку останова (в качестве параметра может быть указан номер строки или название функции) clear − удалить все точки останова в функции continue − продолжить выполнение программы delete − удалить точку останова display − добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы finish − выполнить программу до момента выхода из функции info breakpoints − вывести на экран список используемых точек останова info watchpoints − вывести на экран список используемых контрольных выражений list − вывести на экран исходный код (в качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк) next − выполнить программу пошагово, но без выполнения вызываемых в программе функций print − вывести значение указываемого в качестве параметра выражения run − запуск программы на выполнение set − установить новое значение переменной step − пошаговое выполнение программы watch − установить контрольное выражение, при изменении значения которого программа будет остановлена Для выхода из gdb можно воспользоваться командой quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d. Более подробную информацию по работе с gdb можно получить с помощью команд gdb -h и man gdb.

### *Вопрос 9*

Cхема отладки программы показана в 6 пункте лабораторной работы.

### *Вопрос 10*

При первом запуске компилятор не выдал никаких ошибок, но в коде программы main.c допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке scanf(“%s”, &Operation); нужно убрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.

### *Вопрос 11*

Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: cscope − исследование функций, содержащихся в программе, lint − критическая проверка программ, написанных на языке Си.

### *Вопрос 12*

Утилита splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора C анализатор splint генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работепрограммы, переменные с некорректно заданными значениями и типами и многое другое.

## Вывод

В ходе данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.